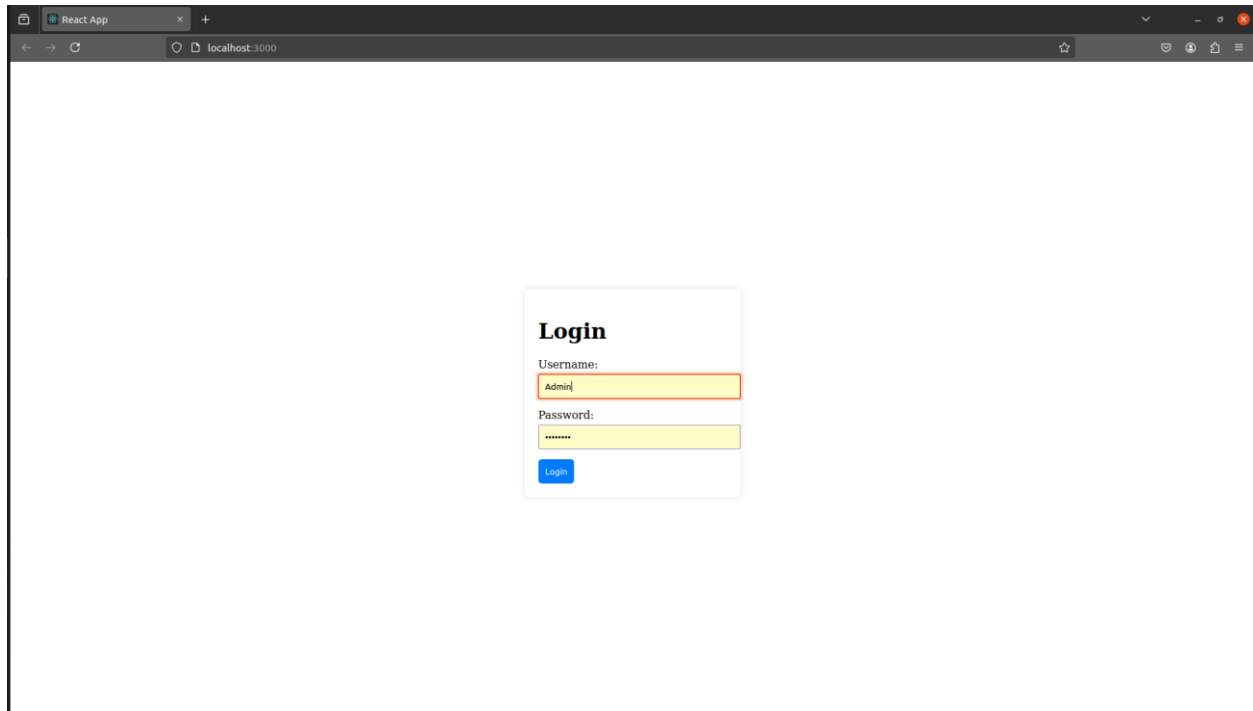


Dashboard Development Assignment Documentation

Git repository: <https://github.com/ShwetaPokale/dashboard>

Frontend:

Login:

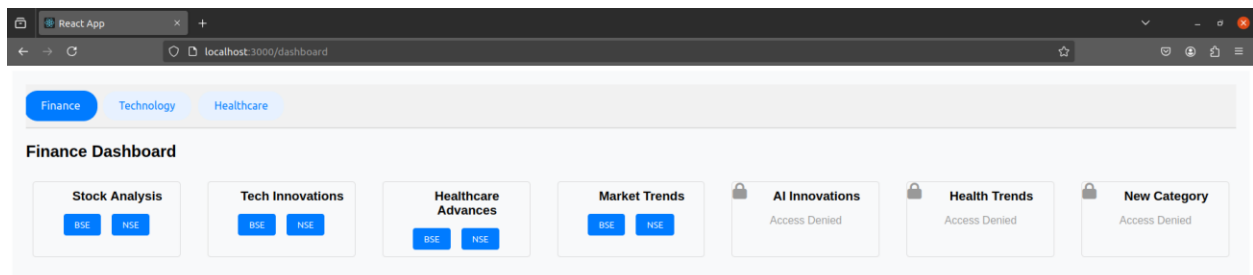


Dashboard

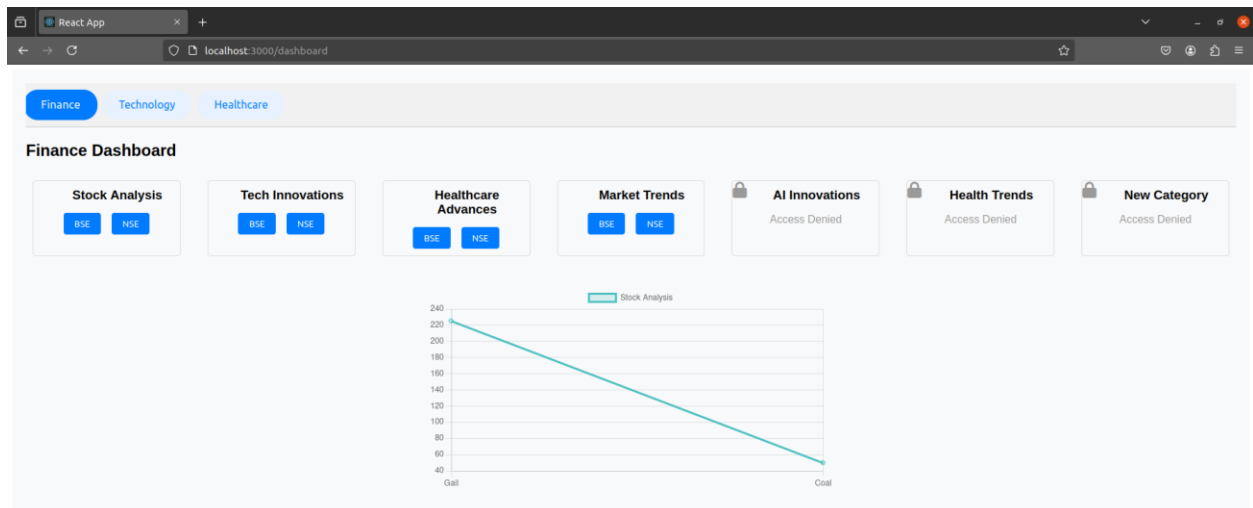
Sections:



Category Page:



Charts:



Backend:

Architecture

- **Framework:** NestJS (Node.js version 20.14, NestJS version 10.4.4)
- **Database:** PostgreSQL
- **ORM:** TypeORM for database management and object-relational mapping.

Database Schema

Tables:

- **users:** Stores user information and credentials.
- **bse_data_source:** Contains data related to the Bombay Stock Exchange (BSE).
- **categories:** Defines various categories for data organization.
- **category_charts:** Associates categories with charts for visualization.
- **charts:** Stores information about charts used in the dashboard.
- **data_sources:** Represents different data sources available in the application.
- **nse_data_source:** Contains data related to the National Stock Exchange (NSE).
- **sections:** Represents different sections in the dashboard.
- **user_categories:** Manages user-specific access to categories.

Backend Setup and API Usage

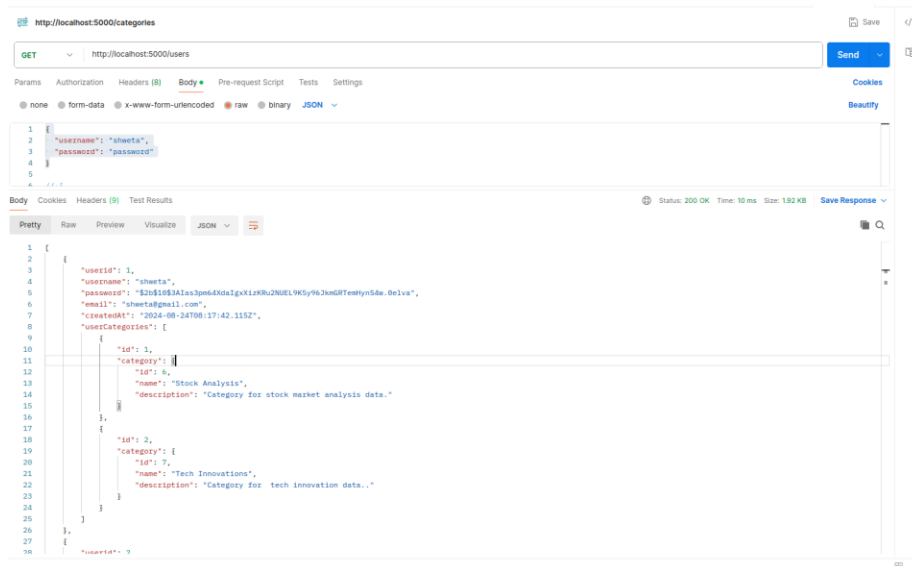
1. **Install Dependencies:**
Path: dashboard-backend
command: npm install
2. **Database Configuration:** Ensure PostgreSQL is configured correctly. Update your database credentials in the configuration file.
Path: dashboard-backend/src/config.ts
3. **Start the Application:** Run the backend server using:
Path: dashboard-backend
command: npm start / npm run start
4. **API Endpoints:**

Users

- **Create User Endpoint:** POST /users
- While creating user can give access to categories based on category id.
- **Request Body:**

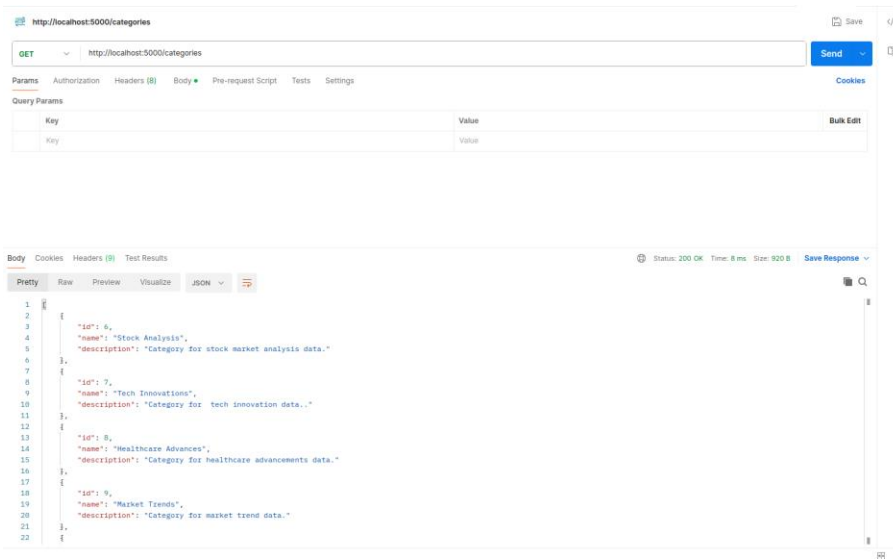
```
{
  "username": "string",
  "password": "string",
  "email": "string",
  "categoryIds": [number]
}
```

- **Get All Users Endpoint:** GET /users
- **Get User by ID Endpoint:** GET /users/:id
- **Update User Endpoint:** PUT /users/:id
- **Delete User Endpoint:** DELETE /users/:id



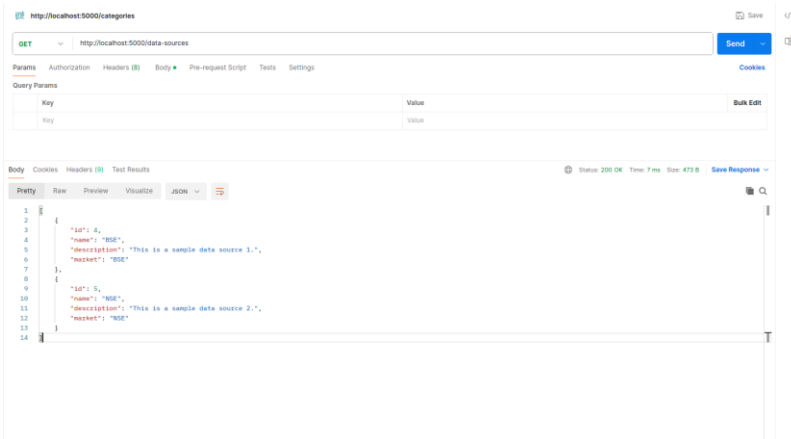
Categories:

- **Get All Categories:** GET /categories
- **Get Categories by User and Section:** GET /categories/user/:userId/section/:sectionId
- **Get Category by ID:** GET /categories/:id
- **Create Category:** POST /categories
- **Update Category:** PUT /categories/:id
- **Delete Category:** DELETE /categories/:id



Data Sources:

- **Create Data Source Endpoint:** POST `/data-sources`
- **Get All Data Sources**
- **Endpoint:** GET `/data-sources`
- **Get Market Data Endpoint:** GET `/data-sources/:type`
 - **Response:** Market data (BSE or NSE).
- **Create Market Data Endpoint:** POST `/data-sources/:type`
- **Update Data Source Endpoint:** PUT `/data-sources/:id`
- **Delete Data Source Endpoint:** DELETE `/data-sources/:id`



The first screenshot shows a GET request to `http://localhost:5000/categories` with a response status of 200 OK. The response body is a JSON array of two category objects. The first object has an id of 1, a name of 'Gall', a close value of 229, a dayHigh of 229, a dayLow of 229, a volume of 3420000, and a percentDelivery of 50. The second object has an id of 2, a name of 'Coal', a close value of 500, a dayHigh of 500, a dayLow of 500, a volume of 9420000, and a percentDelivery of 60.

The second screenshot shows a GET request to `http://localhost:5000/categories` with a response status of 200 OK. The response body is a JSON array of two category objects. The first object has an id of 1, a name of 'Coal', a close value of 500, a dayHigh of 500, a dayLow of 500, a volume of 9420000, and a percentDelivery of 60. The second object has an id of 2, a name of 'Gall', a close value of 200, a dayHigh of 200, a dayLow of 200, a volume of 3420000, and a percentDelivery of 50.

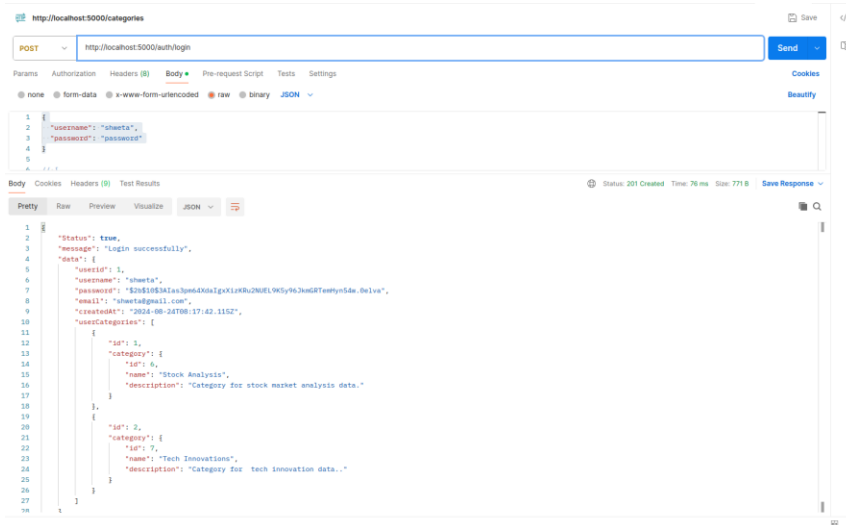
Sections:

- **Get All Sections:** GET /sections
- **Get Section by ID:** GET /sections/:id
- **Create Section:** POST /sections
- **Update Section:** PUT /sections/:id
- **Delete Section:** DELETE /sections/:id

The screenshot shows a GET request to `http://localhost:5000/sections` with a response status of 200 OK. The response body is a JSON array of three section objects. The first object has an id of 1, a name of 'Finance', and a description of 'Section for finance-related data.' The second object has an id of 2, a name of 'Technology', and a description of 'Section for technology-related data.' The third object has an id of 3, a name of 'Healthcare', and a description of 'Section for healthcare-related data.'

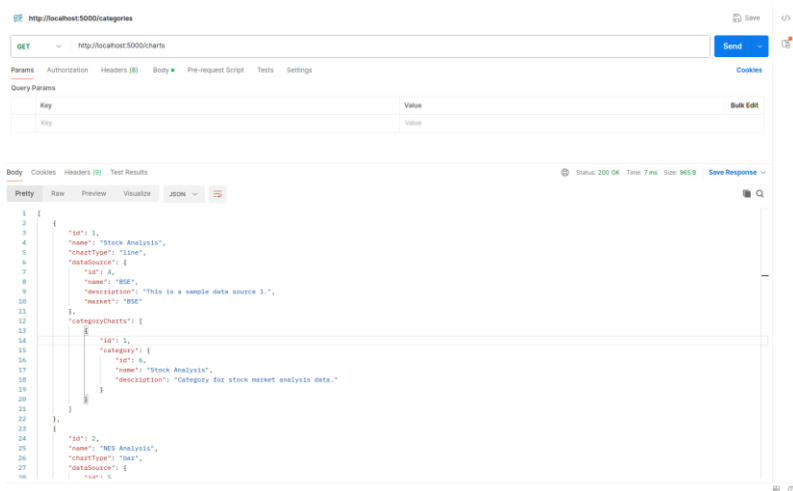
Login

- **Endpoint: POST /auth/login**



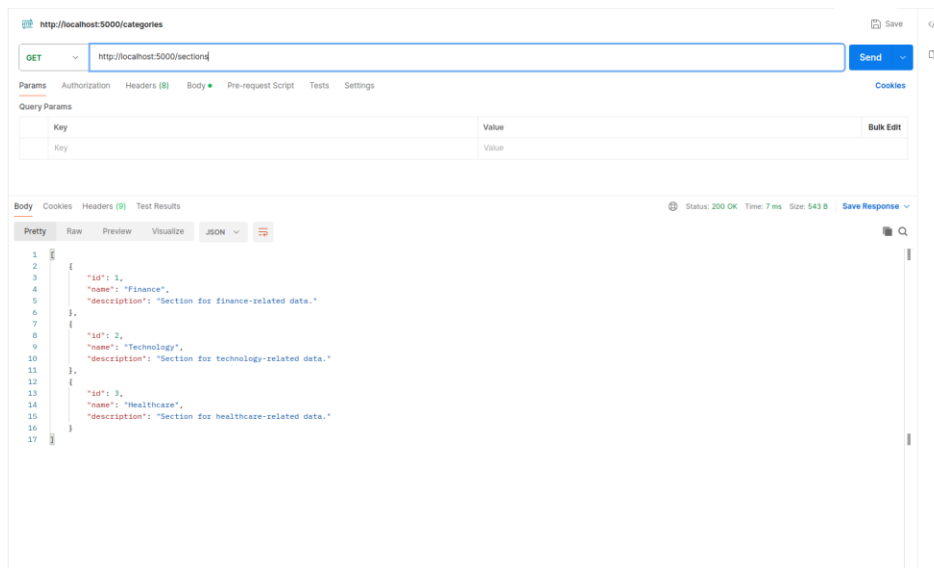
Charts

- **Create Chart Endpoint: POST /charts**
- **Create Category-Chart Endpoint: POST /charts/category-charts**
- **Get All Charts Endpoint: GET /charts**



Sections

- **Create Section Endpoint:** POST /sections
- **Get All Sections Endpoint:** GET /sections
- **Get Section by ID Endpoint:** GET /sections/:id
- **Update Section Endpoint:** PUT /sections/:id
- **Delete Section Endpoint:** DELETE /sections/:id



•

```
shweta@G0T02B3-SHWETA: ~  
shweta@G0T02B3-SHWETA:~$ psql -h localhost -U root -d dashboard  
Password for user root:  
psql (12.20 (Ubuntu 12.20-0ubuntu0.20.04.1))  
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)  
Type "help" for help.  
  
dashboard=> \d  
  
List of relations  
Schema | Name | Type | Owner  
-----+-----+-----+-----  
public | bse_data_source | table | root  
public | bse_data_source_id_seq | sequence | root  
public | categories | table | root  
public | categories_id_seq | sequence | root  
public | category_charts | table | root  
public | category_charts_id_seq | sequence | root  
public | charts | table | root  
public | charts_id_seq | sequence | root  
public | data_sources | table | root  
public | data_sources_id_seq | sequence | root  
public | nse_data_source | table | root  
public | nse_data_source_id_seq | sequence | root  
public | sections | table | root  
public | sections_id_seq | sequence | root  
public | user_categories | table | root  
public | user_categories_id_seq | sequence | root  
public | users | table | root  
public | users_userid_seq | sequence | root  
(18 rows)  
  
dashboard=>
```

5. Unit Test Cases:

- The tests are implemented using Jest and verify the behavior of the login method.

AuthController:

- Scenarios:
 - Username or Password Missing
 - User Does Not Exist
 - Incorrect Password
 - Successful Login

The screenshot shows the Visual Studio Code interface with the file explorer on the left, the source code in the center, and the terminal at the bottom. The terminal displays the command to run Jest tests and the resulting output, which shows that all four test suites passed.

```

auth.controller.spec.ts - dashboard - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
  dashboard-backend
  > dist
  > node_modules
  > src
  > auth
    ts auth.controller.spec.ts
    ts auth.controllers
    ts auth.module.ts
  > categories
  > charts
  > datasource
  > sections
  > users
  ts app.controller.spec.ts
  ts app.controllers
  ts app.module.ts
  ts app.service.ts
  ts main.ts
  .eslintrc.js
  .gitignore
  .prettierrc
  () nest-cli.json
  () package-lock.json
  () package.json
  () README.md
  () tsconfig.build.json
  tsconfig.json
  > dashboard-frontend

ts auth.controller.spec.ts
dashboard-backend > src > auth > ts auth.controller.spec.ts > ...
6 describe('AuthController', () => {
30 describe('login', () => {
37 it('should return an error if the user does not exist', async () => {
38 mockUsersService.findByUsername.mockResolvedValue(null);
39 const result = await authController.login({ username: 'user', password: 'pass' });
40 expect(result).toEqual({ Status: false, message: 'Invalid username or password' });
41 });
42 // scenario 3: Wrong password
43 it('should return an error if the password does not match', async () => {
44 mockUsersService.findByUsername.mockResolvedValue({ userid: 1, password: 'hashedPassword' });
45 jest.spyOn(bcrypt, 'compare').mockResolvedValue(false);
46 const result = await authController.login({ username: 'shweta', password: 'abcd' });
47 expect(result).toEqual({ Status: false, message: 'Invalid username or password' });
48 });
49 // scenario 4: Username and password is correct
50 it('should return user data on successful login', async () => {
51 mockUsersService.findByUsername.mockResolvedValue({ userid: 1, password: 'hashedPassword' });
52 jest.spyOn(bcrypt, 'compare').mockResolvedValue(true);
53 mockUsersService.findOne.mockResolvedValue({ userid: 2, username: 'user' });
54 const result = await authController.login({ username: 'shweta', password: 'password' });
55 expect(result).toMatchObject({ Status: true, message: 'Login successfully' });
56 });
57 });
58 });
59 });

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

shweta@G0782B3-SHWETA:~/Documents/dashboard/dashboard-backend$ npx jest src/auth/auth.controller.spec.ts
PASS src/auth/auth.controller.spec.ts
  AuthController
    login
      ✓ should return an error if username or password is missing (11 ms)
      ✓ should return an error if the user does not exist (3 ms)
      ✓ should return an error if the password does not match (2 ms)
      ✓ should return user data on successful login (3 ms)

Test Suites: 1 passed, 1 total
Tests: 4 passed, 4 total
Snapshots: 0 total
Time: 2.771 s, estimated 3 s
Ran all test suites matching /src/auth/auth.controller.spec.ts/i.
shweta@G0782B3-SHWETA:~/Documents/dashboard/dashboard-backend$
  
```

CategoriesController

- Scenarios
 - Create Category Successfully
 - Missing Category Name
 - Missing Section ID
 - Retrieve All Categories
 - Retrieve Category by ID
 - Missing User ID When Retrieving Category
 - No Access to Category
 - Update Category Successfully
 - No Fields Provided for Update
 - Delete Category Successfully

- Category Not Found During Deletion
- Find Categories by User and Section

The screenshot shows the Visual Studio Code interface with the following components:

- Explorer:** Shows the project structure with folders like `dashboard-backend`, `src`, and `categories`.
- Editor:** Displays the `categories.controller.ts` file. The code includes:


```

import { UsersService } from '../users/users.service';

@Controller('categories')
export class CategoriesController {
  constructor(private readonly categoriesService: CategoriesService,
    private readonly userService: UsersService,
  ) {}

  @Post()
  async create(@Body() body: any): Promise<Category | string> {
    const { name, section, description } = body;
    if (!name) {
      return 'Name is required';
    }
    else if (!section || !section.id) {
      return 'Section with ID is required';
    }
    const category = { name, section, description };
    return this.categoriesService.create(category);
  }

  @Get()
  async findAll(): Promise<Category[]> {
    return this.categoriesService.findAll();
  }
}

```
- Terminal:** Shows the command `npm test` and its output. The output indicates that all tests passed:


```

Run all test suites matching /src/categories/categories.controller.spec.ts/
PASS src/categories/categories.controller.spec.ts
CategoriesController
  ✓ should create a category (12 ms)
  ✓ should return error if name is missing when creating a category (2 ms)
  ✓ should return error if section is missing when creating a category (1 ms)
  ✓ should return all categories (2 ms)
  ✓ should return a single category by ID (4 ms)
  ✓ should return error if user ID is missing when finding a category (2 ms)
  ✓ should return error if user does not have access to the category (2 ms)
  ✓ should update a category (2 ms)
  ✓ should return error if no fields are provided for update (2 ms)
  ✓ should delete a category (2 ms)
  ✓ should return error if category not found during deletion (2 ms)
  ✓ should find categories by user and section (3 ms)

Test Suites: 1 passed, 1 total
Tests: 12 passed, 12 total
Snapshots: 0 total
Time: 2.957 s, estimated 3 s

```

6. Static Code Quality Report

- Report is generated using ESLint

ESLint Report link:

<https://github.com/ShwetaPokale/dashboard/blob/master/dashboard-backend/eslint-report.xml>